TECHNICAL REPORT RD-GC-92-26

(2)

A FUZZY LOGIC CONTROLLER SYSTEM

G. Patton Bradford
Guidance and Control Directorate
Research, Development, and Engineering Center

NOVEMBER 1992

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5000

Approved for public release;distribution is unlimited.

DESTRUCTION NOTICE

FOR CLASSIFIED DOCUMENTS, FOLLOW THE PROCEDURES IN DoD 5200.22-M, INDUSTRIAL SECURITY MANUAL, SECTION II-19 OR DoD 5200.1-R, INFORMATION SECURITY PROGRAM REGULATION, CHAPTER IX. FOR UNCLASSIFIED, LIMITED DOCUMENTS, DESTROY BY ANY METHOD THAT WILL PREVENT DISCLOSURE OF CONTENTS OR RECONSTRUCTION OF THE DOCUMENT.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL ENDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188 Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) TR-RD-GC-92-26	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Guidance and Control Directorate RD&E Center 6b. OFFICE SYMBOL (If applicable) AMSMI-RD-GC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Commander, U.S. Army Missile Command ATTN: AMSMI-RD-GC-S Redstone Arsenal, AL 35898-5254	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) A Fuzzy Logic Controller System	
12. PERSONAL AUTHOR(S) G. Patton Bradford	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM _____ TO Feb 92	14. DATE OF REPORT (Year, Month, Day) November 1992 15. PAGE COUNT
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Fuzzy Logic, Control Systems, Expert Systems, Fuzzy Set Theory

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Fuzzy set theory allows for a relatively straightforward method of working with uncertain or imprecise data. Fuzzy logic makes use of this theory by providing a methodology of applying rules to this data.

Presented in this paper is a short discussion of fuzzy set theory, followed by a discussion of the system simulated. In the work discussed in this paper, a simple feedback system is used to illustrate the application of fuzzy logic. The system simulated is a simplified model of a turret whose position is to be controlled. The results of the simulation are shown and elaborated on, followed by a discussion of potential applications for fuzzy controllers and expert systems. As an appendix, the computer source code used to implement the simulation is presented and discussed.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL G. Patton Bradford	22b. TELEPHONE (Include Area Code) (205) 876-7081 22c. OFFICE SYMBOL AMSMI-RD-GC-S

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

SUMMARY

Fuzzy set theory, developed by Dr. Lotfi Zadeh in the mid–1960's, allows for a relatively straightforward method of working with uncertain or imprecise data. Fuzzy logic makes use of this theory by providing a methodology of applying rules to this data.

In the work discussed in this paper, a simple feedback system is used to illustrate the application of fuzzy logic. The system simulated is a simplified model of a turret whose position is to be controlled. The inputs to the fuzzy controller would be the difference in the commanded position and the present position, and the present angular velocity. The output of the controller is a change in the angular velocity of the turret.

Presented in this paper is a short discussion of fuzzy set theory, followed by a discussion of the system simulated. The fuzzy rule set and the allocation of values to the fuzzy sets is presented. The results of the simulation are shown and elaborated on, followed by a discussion of potential applications for fuzzy controllers and expert systems. Appendix A discusses the computer source code used to implement the simulation.

PREFACE

Credit for the possibility of using fuzzy logic for this application and for the other possible applications listed in this document rightfully goes to my coworker, Ms. Wanda Hughes. I am indebted to her for her contribution to this work. Additionally, thanks is extended to Mr. John Carter and Mr. Michael Pitruzzello for their help and input in familiarizing me with the problems in the area of Forward Area Air Defense.

The discussion of the fundamentals of fuzzy logic is taken from Dr. Bart Kosko's excellent work, Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence, Prentice–Hall, 1991. The reader is referred to this work if a more indepth discussion of fuzzy set theory is desired.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

One of the more pressing problems in the air defense arena is the problem of fratricide. In a complex battlefield, how does one separate friend from foe? When there are multiple source of information available, some of which are imprecise, uncertain, or even contradictory, how does one get an accurate picture of the battlefield? Though fratricide reduction and control of a turret seem to have nothing in common, it is this first problem which precipitated the work which is the basis for this report.

One of the systems supported by this office is the AVENGER air defense system. Presently, efforts are underway to integrate it into the Forward Area Air Defense Command and Control (FAAD–C2) network in order to provide individual AVENGER units with cuing information. Additionally, it is proposed that other, more local, sources of information, such as Noncooperative Target Recognition (NCTR), also be used by these units. The problem is that neither targeting system is complete, each with its own set of strengths and weaknesses. While FAAD–C2 provides accurate positional data and less accurate identification data, NCTR is just the opposite with incomplete and imprecise positional data and identification data of high accuracy. What is desired is a system which can fuse these data sources to provide a situational picture more accurate than either source can provide individually.

In order to accomplish this fusion in an effective and efficient manner, this office has undertaken an investigation of some nontraditional techniques which appear to perform well in environments of uncertainty. Two that have been considered in particular are neural networks and fuzzy logic. This document reports on some first efforts in the latter.

When first proposed as a possible solution, there was no existing expertise in fuzzy logic in this office. In order to develop the required knowledge, a first problem was chosen which would allow relatively fast development of functioning computer code and an obvious way of checking the results. Control of a physical system fit this description well; also, automatic slewing of the turret to the target of greatest threat has been proposed, so the results could be directly applicable.

II. FUZZY SET THEORY AND FUZZY LOGIC

A. Fuzzy Set Theory

Fuzzy set theory is not a new subject; it was originally developed by Dr. Lotfi Zadeh in the mid–1960's. However, in our modern world with its bias toward the binary, it is unfamiliar to most people. What follows is a very rudimentary discussion of fuzzy set theory and the system of fuzzy logic associated with it.

Consider our traditional view of sets. Something is either an element of a set, or it is not; there is no in between. For instance, if a set was the set of all persons over 1.8 m tall, one could determine precisely who would and who would not be in the set. There would not be someone who is both over 1.8 m tall and under 1.8 m tall.

Not all descriptions are like this, however. Consider the set of all "tall" people. What is "tall?" Now we get into an area where there are instances of the class which represent the class better than other instances. For instance, someone 2 m tall would represent the class of "tall" people better than someone 1.7 m tall, though the latter could still be considered "somewhat" tall. Additionally, these two persons may be in other sets, also, such as the set of "very tall" persons and that of "medium height" persons, again to different degrees.

Fuzzy set theory attempts to quantify this degree of membership aspect of sets whose descriptors tend to be linguistic in nature ("big," "little," "most," "somewhat,"etc.). Figure 1 shows how this might be done.

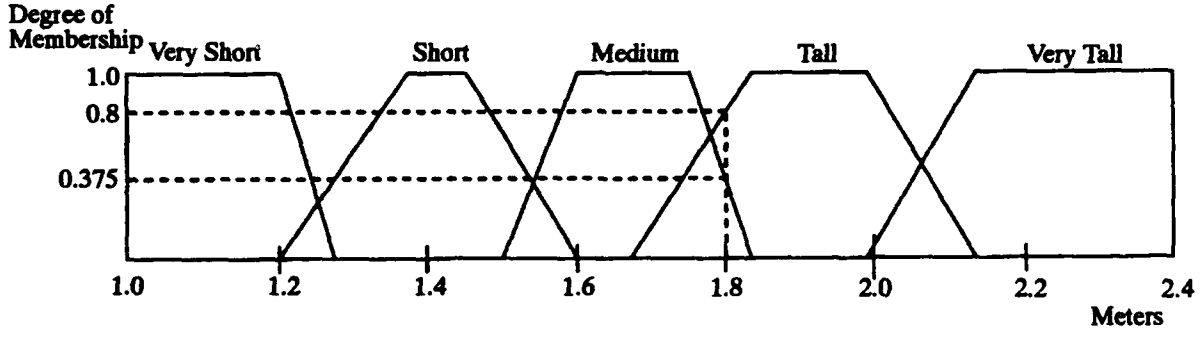

Figure 1. Fuzzy Set Membership Functions

Each of the trapezoids in the figure defines a membership function for each of the fuzzy sets. These functions define the degree of membership between 0 and 1 of a value within a particular set. For instance, for the "Very Short" set, any value under 1.2 m is a part of that set with degree 1. Above that point, membership in this set decreases linearly until it reaches zero at about 1.3 m. Important to note is that a value can be a member of two distinct sets at the same time. For instance, if a person were 1.8 m tall, with this set definition, he would be in the set "Medium Height" with degree 0.375, in "Tall" with degree 0.8, and in all other sets with degree zero.

The definitions of the union and intersection of fuzzy sets are a very straightforward and easily understood: the degree of membership of an element in the union of two sets is the maximum of the degree of membership in either set, and the degree of membership of an

element in the intersection of two sets is the minimum of the degree of membership in either set. For instance, if an element were in one set to degree 0.3 and in another set to degree 0.7, it would be in the union to degree 0.7 and in the intersection to degree 0.3. As a second example, if an element were in one set to degree 0.5 and in another set to degree zero, it would be in the union to degree 0.5 and in the intersection to degree zero. This is in agreement with the usual definitions of union and intersection in that, if an element is in either set, it is in the union, and if it is not in both sets, it is not in the intersection of the sets.

The extension to logic, naturally, is direct. One must simply think of the degree of truth as the degree to which an element is a member of a set for which some condition holds true. This being the case, AND is the minimum of truth values (the same as intersection), OR is the maximum of truth values (the same as union), and NOT is 1 minus the truth value. For an implication (an IF ... THEN statement), the truth value of the conclusion is the same as the truth value of the proposition. While this is very natural, in the extreme cases of where the degree of membership is one or zero (the realm of traditional Boolean logic), it leads to a significant deviation from the results of Boolean logic: if the proposition is of degree zero, the conclusion is also of degree zero, not degree one as in Boolean logic. This property of fuzzy logic alleviates the problem in Boolean logic of the false proposition where, if the proposition is false, the implication is considered true, no matter what the conclusion.

B. Fuzzy Expert Systems

Given the logic system defined above, it is possible to implement a computationally efficient fuzzy expert system for making decisions or controlling other systems. An example is probably the best way of explaining how such a system might be implemented. Consider a system defined by Figure 2 and Table 1. The first drawing in Figure 2 shows the membership functions for the first input; the second drawing shows the membership functions for the second input; and the third drawing shows the membership functions for the output. (The reason for using symmetric triangular and trapezoidal functions will be explained later). The set memberships are range as follows:

Input 1:

- The ZERO set runs from 0 to 7, centered about 3.5.
- The SMALL set runs from 5 to 11, centered about 8.
- The MEDIUM set runs from 10 to 16, centered about 13.
- The LARGE set runs from 14 to 20, centered about 17.

Input 2:

- The ZERO set runs from 0 to 12, centered about 6.
- The SMALL set runs from 8 to 24, centered about 16.
- The MEDIUM set runs from 18 to 30, centered about 24.
- The LARGE set runs from 24 to 40, centered about 32.

Output:

- The ZERO set runs from 0 to 25, centered about 12.5.
- The SMALL set runs from 15 to 60, centered about 37.5.
- The MEDIUM set runs from 50 to 75, centered about 62.5.
- The LARGE set runs from 70 to 100, centered about 85.

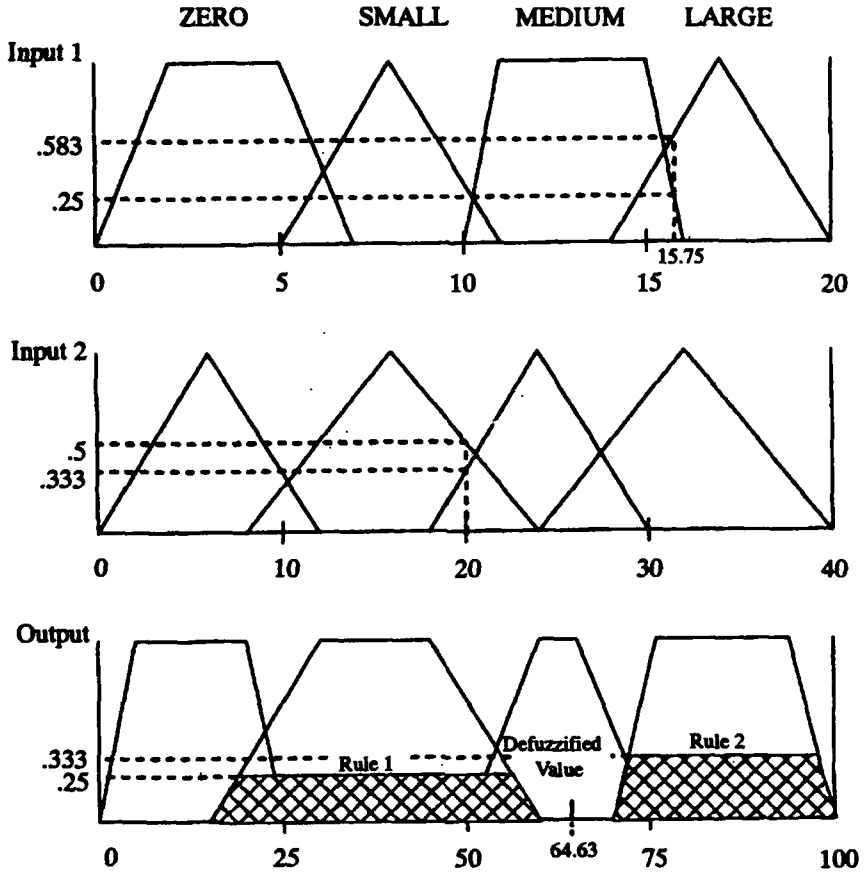

Figure 2. Example Membership Functions

Table 1. Example Rule Set

		Input 2			
		Zero	**Small**	**Medium**	**Large**
Input 1	**Zero**	Small	Zero	Medium	Small
	Small	Zero	Medium	Small	Medium
	Medium	Large	Small	Medium	Zero
	Large	Medium	Large	Large	Large

Table 1 gives the rule base for this system. This table is interpreted in this manner: the row and column names represent the ANDed propositions (inputs) to each rule, and each entry is the conclusion (output) for each of those propositions. For instance, the first entry into the table would represent the rule, "IF Input 1 is Zero AND Input 2 is Zero, THEN the Output is Small."

To show how this expert system would work, consider the following situation. Input 1 to the system is 15.75 and Input 2 is 20. If the rules are invoked, what should the Output value be? For the sake of simplicity, we shall only invoke two rules:

1) IF Input l is Medium AND Input 2 is Small THEN Output is Small; and

2) IF Input 1 is Large AND Input 2 is Medium THEN Output is Large.

Consider first the propositions of Rule 1. From Figure 2 and the description above, it can be seen that the first statement, "Input 1 is Medium," is true to degree 0.25, and the second statement, "Input 2 is Small," is true to degree 0.5. Since the value of the AND condition would be the minimum of these two values, the value of the proposition is 0.25. Therefore, the value of the conclusion of Rule 1 would also be 0.25; i.e., "Output is Small" is true to degree 0.25. Rule 2 is tested the same way. "Input 1 is Large" is true to degree 0.583, and "Input 2 is Medium" is true to degree 0.333 (Note how seemingly contradictory data is being handled in this system.). Therefore, the proposition of rule 2 is true to degree 0.333, and, according to this rule, "Output is Large" is true to degree 0.333.

C. Defuzzification

It would seem that the rule set has produced contradictory results; how are these to be reconciled? Even if only one result were given and there were no contradictions, the output of a rule is not a distinct value, but a degree of membership within a set. Since most systems would need one single value on which to operate, how is this value obtained? Both of these questions are addressed in the process of "defuzzification." The method used in the presented work in the centroidal defuzzification technique described by Kosko in his book, Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence. Consider this: the output from each rule can be considered a "fit vector" whose elements represent the degree of membership in each of the output's fuzzy sets. This method works with the global centroid of the rule results from the local centroids of each set weighted by their fit values. This is described by Kosko with the function:

$$V_k = \frac{\sum_{j=1}^{n} m_o(y_j) y_j}{\sum_{j=1}^{n} m_o(y_j)}$$

where v_k is the centroid value of the fit vectors, n is the number of rules, $m_o(y)$ is the degree of membership in each set, and y_j is the centroid of that set.

For the example problem given above, the fit vectors would be (0, .25, 0, 0) and (0, 0, 0, .333), respectively. Therefore, the summed output vector would be (0, .25, 0, .333). With the symmetric trapezoid–like functions used in the example, the centroids of the sets are simply the central values, and the vector of set centroids is (12.5, 37.5, 62.5, 85). Based on these values and the above equation, the defuzzified result would be:

$$\frac{(0 \times 12.5) + (.25 \times 37.5) + (0 \times 62.5) + (.333 \times 85)}{.25 + .333}$$

or 64.63, squarely in the Medium set of the output.

There are two important points to note on fuzzy systems. First of all, it provides a straightforward mechanism for handling contradictory system constraints. Second, it is computationally efficient. There is only one division in the entire process, all other operations being either addition, multiplication, or comparison. For computer implementation, some computations, such as the determinations of the centroid of the fuzzy sets, can be handled through a table look–up.

III. PROBLEM DESCRIPTION

A. Turret Slewing for Controller Development

As stated previously, the development of a fuzzy controller for turret slewing was not an end in itself; it was used simply as a convenient vehicle for developing the framework for a more generic fuzzy expert system. The simulation of the turret motion, therefore, will be very simple: A maximum slewing rate of one revolution per second and a controller update rate of one tenth of a second are assumed. It is also assumed that, if the turret is stationary, it takes two cycles for any change in commanded slew rate to take effect. This means that half of the commanded change in rate is added on any given cycle. This is a very simplistic approach to turret motion, but it serves the purposes of this study. There was one constraint, however, that was put on the motion of the turret: since there would be a man in the turret, there should be no sudden accelerations which would cause excessive jarring.

B. Controller Description

The controller for the turret was implemented as a fuzzy expert system, controlling the velocity of the turret with respect to the present position and the present angular rate. The inputs to the fuzzy controller are 1) the difference in the commanded turret position and the present position (DeltaPosition), and 2) the present angular rate of the turret (Present–Rate). The output of the controller is a change in the angular rate (DeltaRate).

1. Fuzzy Set Membership Functions

The fuzzy sets were defined as follows:

DeltaPosition:

LargeNegative – A trapezoidal function with endpoints at –180 and –105 with roll–off points at –170 and –115.

MediumNegative – A triangular function with endpoints at –123.75 and –48.75 and a maximum at –86.25.

SmallNegative – A triangular function with endpoints at –67.5 and 0 and a maximum at –33.75.

Zero – A triangular function with endpoints at –20 and 20 and a maximum at 0.

SmallPositive – A triangular function with endpoints at 0 and 67.5 and a maximum at 33.75.

MediumPositive – A triangular function with endpoints at 48.5 and 123.75 and a maximum at 86.25.

LargePositive – A trapezoidal function with endpoints at 105 and 180 with roll–off points at 115 and 170.

PresentRate:

LargeNegative – A trapezoidal function with endpoints at –36 and –21 with roll–off points at –34 and –23.

MediumNegative – A triangular function with endpoints at –24.75 and –9.75 and a maximum at –17.25.

SmallNegative – A triangular function with endpoints at –13.5 and 0 and a maximum at –6.75.

Zero – A triangular function with endpoints at –4 and 4 and a maximum at 0.

SmallPositive – A triangular function with endpoints at 0 and 13.5 and a maximum at 6.75.

MediumPositive – A triangular function with endpoints at 9.75 and 24.75 and a maximum at 17.25.

LargePositive – A trapezoidal function with endpoints at 21 and 36 with roll–off points at 23 and 34.

DeltaRate:

LargeNegative – A trapezoidal function with endpoints at –18 and –10.5 with roll–off points at –17 and –11.5.

MediumNegative – A triangular function with endpoints at –12.375 and –4.875 and a maximum at –8.625.

SmallNegative – A triangular function with endpoints at –6.75 and 0 and a maximum at –3.375.

Zero – A triangular function with endpoints at –2 and 2 and a maximum at 0.

SmallPositive – A triangular function with endpoints at 0 and 6.75 and a maximum at 3.375.

MediumPositive – A triangular function with endpoints at 4.875 and 12.375 and a maximum at 8.625.

LargePositive – A trapezoidal function with endpoints at 10.5 and 18 with roll–off points at 11.5 and 17.

These are shown graphically in Figure 3.

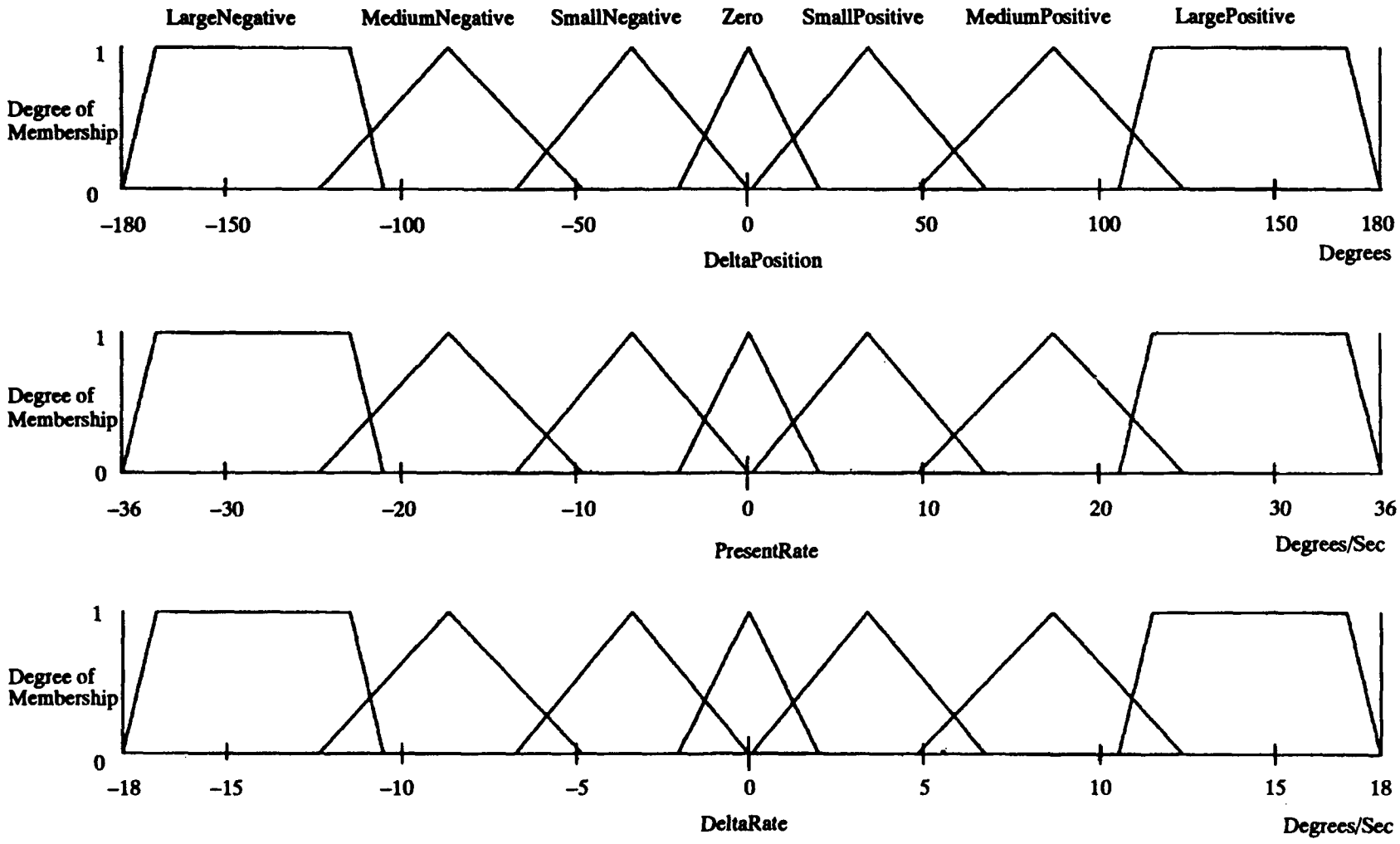

Figure 3. Controller Set Membership Functions

Originally, SmallNegative and Smallpositive did not cover all the way to 0.0. However, on a test run, it was found that if the turret is stationary and the only rule invoked resulted in any degree of Zero as a conclusion, nothing would happen, and the system would be stuck. With the SmallNegative and SmallPositive extending to 0.0, this possibility is eliminated.

2. Fuzzy Rule Base

The first proposed rule set is shown in Table 2. The row values shown in each column are the DeltaRate for the given DeltaPosition and PresentRate. Note that this table list all of the possible inputs and proposed outputs are listed; only those marked were actually implemented in the code. The abbreviations mean: LN – Large Negative, MN – Medium Negative, SN – Small Negative, ZE – Zero, SP – Small Positive, MP – Medium Positive, LP – Large Positive.

Table 2. Rule Set for First Run

		PresentRate						
		LN	**MN**	**SN**	**ZE**	**SP**	**MP**	**LP**
DeltaPosition								
	LN	ZE+	SN	MN	SN*	LN	LN	LN*
	MN	SP	ZE	SN*	SN*	LN*	LN*	LN
	SN	MP	SP	ZE*	SN*	MN*	LN	LN
	ZE	LP	MP	SP*	ZE*	SN*	MN	LN
	SP	LP	LP	MP*	SP*	ZE*	SN	MN
	MP	LP	LP*	LP*	SP*	SP*	ZE	SN
	LP	LP*	LP	LP	SP*	MP	SP	ZE

+Table values represent the change in angular rate (DeltaRate) for the given PresentRate and DeltaPosition.

*Implemented for the first run.

IV. RESULTS

With an initial turret position of 0°, the simulation was run with the commanded positions shown in Table 3. The number of commands was kept purposely small so that the resulting graphs would be uncluttered and there would be a smaller result file to handle. However, the commands given are representative in that they include small, medium, large, positive, and negative changes in position.

Table 3. Commanded Turret Positions

–12.5
–15.0
10.0
12.0
180.0
0.0

Figure 4 shows the resulting performance of the system. Note that for small changes in commanded position, the system behaves as if it were overdamped, gradually sliding into the commanded position; however, for large changes, the system is somewhat underdamped. Two points are of interest here. First of all, though the overshoot was over 30 percent, there was no ringing about the final position. Second, with the large changes in position, the system would start out with some velocity (6.75° per sec), stay there for about a second and a half, and then go to double that velocity (13.5° per sec) in two tenths of a second and stay there for the rest of the climb. This latter show the system coming out from under the effects of the rules used to maintain the system at zero delta position. The former shows the effect on the system of coming back under them.

This amount of overshoot would be considered undesirable in a turret system, so ways of reducing it or eliminating it were investigated. This is one of the beauties of fuzzy systems: significant changes to system performance can be accomplished through relatively simple changes to either the rule base or the set memberships. Because of its simplicity, it was decided just to look at the effects of rule changes.

In looking at the implemented rule set, it was noted that no rules covered the situation where the DeltaPosition is small, but the PresentVelocity is Large in the direction that would reduce DeltaPosition. In order to handle these situations, the changes shown below were made to the rule set. (The rules which were removed were removed just to keep the number of rules constant).

Figure 5 shows the results of running the simulation again, but with the slightly modified rule base. In this case, there is no overshoot, and the simulation behaves as an underdamped system for all of the commands. Also note, the velocity on the large DeltaPositions remains constant at 6.75 for most of the sudden change in velocity after the velocity reaches a certain point. This is also a result of the new rules. However, even with this lower velocity, because of lack of overshoot, the total time to settle at the desired position is still about the same.

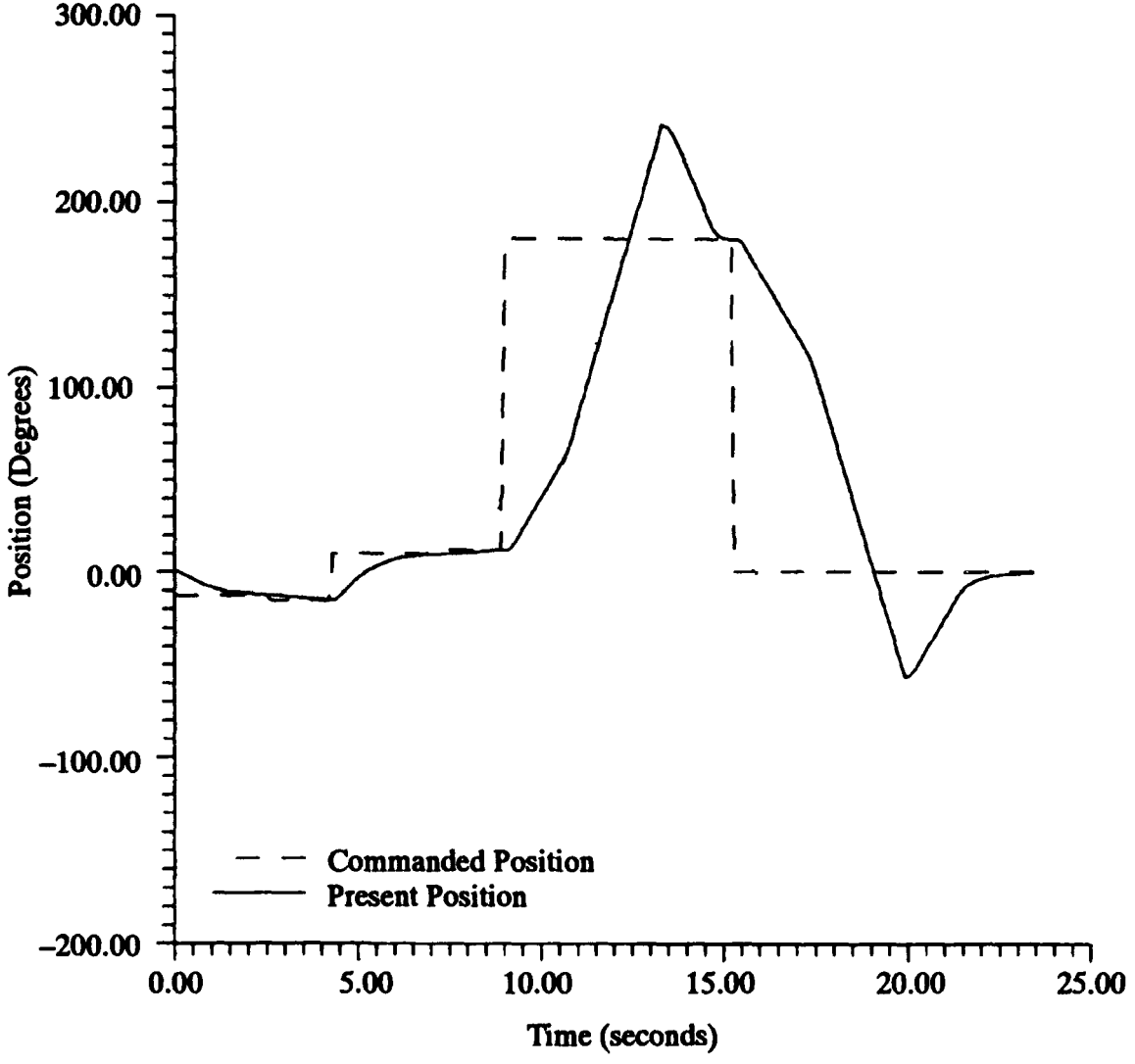

Figure 4. Run 1 Results

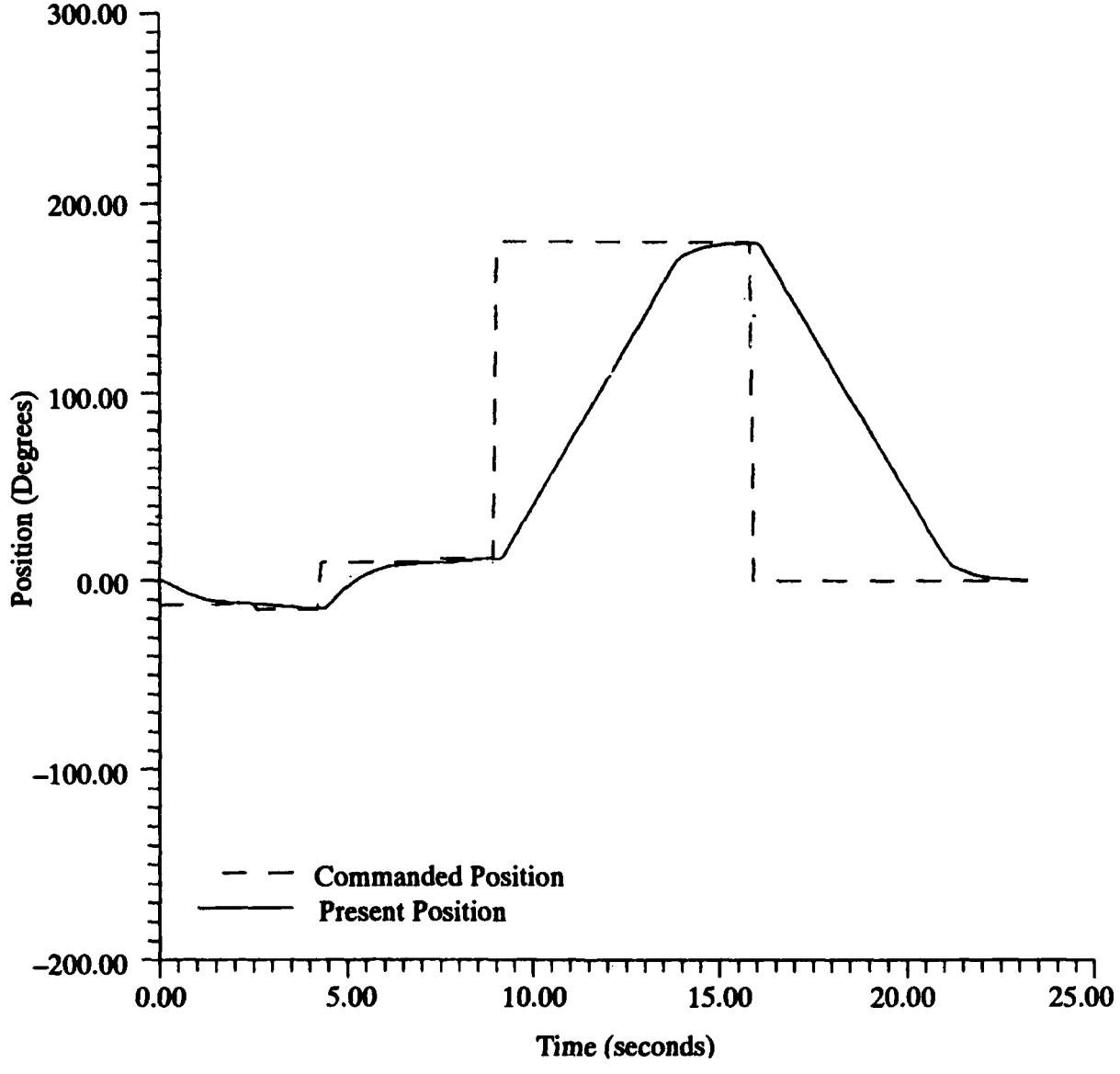

Figure 5. Run 2 Results

Table 4. Rule Set for Second Run

		PresentRate						
		LN	MN	SN	ZE	SP	MP	LP
DeltaPosition								
	LN	ZE	SN	MN	SN*	LN	LN	LN*
	MN	SP	ZE	SN[1]	SN*	LN[1]	LN*	LN
	SN	MP[2]	SP[2]	ZE*	SN*	MN*	LN	LN
	ZE	LP	MP	SP*	ZE*	SN*	MN	LN
	SP	LP	LP	MP*	SP*	ZE*	SN[2]	MN[2]
	MP	LP	LP*	LP[1]	SP*	SP[1]	ZE	SN
	LP	LP*	LP	LP	SP*	MP	SP	ZE

*Used in both runs 1 and 2.

[1]Used in run 1, but not in run 2.

[2]Used in run 2, but not in run 1.

V. CONCLUSIONS

The primary purpose of this study was not to develop a realizable control for a turret system; it was to develop a fuzzy inference engine and illustrate its use in a system familiar to most readers of this report. We believe this has been accomplished. On the other hand, this report does serve to illustrate that fuzzy logic systems would, indeed, be viable as controllers, particularly where system nonlinearities would make implementation of other techniques difficult. Changes in system performance are readily and easily made through simple, straightforward changes in either membership functions or the rule base. For the purposes of the work with which the author is involved, fuzzy logic shows great potential in the evaluation of uncertain data.

APPENDIX A
SIMULATION SOURCE CODE

APPENDIX A
SIMULATION SOURCE CODE

A.1 DISCUSSION

In examining this program, the reader must keep in mind that the original purpose of writing this program was to develop a fuzzy inference engine, something that could be used for many different applications, not something that is optimized for a single application. Also, the program was constructed so that a scripting language for automatically generating the required program files could be developed. Therefore, there will be inefficiencies or points of confusion that will appear in the code that would not be there otherwise. As they come up, this discussion will attempt to clarify these points.

The simulation program consists of three main files: TURRET.C, the driver program; FUZZY.C, the code responsible for determining and maintaining the membership relations on incoming data; and RULES.C, which maintains the rule base, does the inferencing based on the membership values supplied by FUZZY.C, and returns a defuzzified value. Each of these sections of code will be discussed in turn.

TURRET.C, as stated above, is the driver program; this is where the specific problem is coded. Any calls to the outside world are made through here. The only interfaces to the fuzzy controller are through two function calls, **void init_fuzzy (void)** and **float fuzzy (float, float)**. Init_fuzzy( ) initializes the fuzzy logic system (this will be explained shortly), and fuzzy( ) will take the DeltaPosition and the PresentRate and return a value for DeltaRate.

FUZZY.C maintains and performs membership evaluation on the incoming data. In fact, most of the file consists of the various membership functions. There are a couple of enumeration types and several arrays that bear discussion at this point. The enumeration type **FuzzyVariables** lists the inputs and outputs of this problem, i.e., DeltaPosition, PresentRate, and DeltaRate. **FuzzySets** lists the linguistic set descriptors on these three: LargeNegative, MediumNegative, SmallNegative, Zero, SmallPositive, MediumPositive, and LargePositive.

The arrays that bear some attention are **float (* mf_DeltaPosition [MAX_SETS]) (float), float (* mf_PresentRate [MAX_SETS]) (float), float (*mf_DeltaRate [MAX_SETS]) (float),** and **float (*(*MembershipFunction [MAX_VARIABLES])) (float)**. Note that the first three are arrays of function pointers, and the third is an array of pointers to function pointers. These arrays are initialized in init_fuzzy( ), with the first three arrays receiving the addresses of their particular membership functions and MembershipFunction[ ] receiving the addresses of these other three arrays. This is part of what makes this inference engine generic: one does not have to know the name of a membership functions in order to call it. The program, MembershipFunction[ ] is indexed by a variable of type Fuzzy–Variables, and the other three arrays are indexed by FuzzySets1. In this way, if one wished to test the membership in the set MediumNegative of PresentRate of a value x, one would merely have to make a call of the form:
y = (*(*Membership Function[PresentRate][MediumNegative]))(x)

The function fuzzy( ) takes two values, one for the DeltaPosition and the other for PresentRate, and returns a value for DeltaRate. In order to do this, it steps through all of the membership functions with the values of the inputs and saves the results in a global array **InputMemberships[ ][ ]**. The decision to do this was a design decision based on efficiency. One could just as well postponed the computation of set membership until some set was explicitly called from the rules, thus saving the memory reguired to store this information. For instance, if a rule had stated, "IF LargeNegative AND MediumPositive THEN LargeNegative," for input values of x and y, the program could have invoked (* (*MembershipFunction [DeltaPosition] [LargeNegative])) (x) and compared it with the result of (*(*MembershipFunction [PresentRate] [LargePositive])) (y) to get a membership value for LargeNegative on DeltaRate. However, with the twenty–two rules used, each with two ANDed propositions, this would resulted in forty–four function invocations. Computing all of the membership values up front results in only fourteen function invocations.

Once all of the membership values have been computed, fuzzy( ) calls the two functions in RULES.C, **void rules(void)**, and **float defuzzifier(void)**. Associated with this file, also, are two static structures, **RuleSet [NumberofRules]** and **OutputMembership [NumberofRules]**. RuleSet[ ] is a static array of structures, each structure containing three values of type enum FuzzySets1, representing the two propositions and one conclusion for each rule. OutputMembership[ ] is also a static array of structures, one value, representing the conclusion of a rule, being of type enum FuzzySets1, and the other, representing the degree of membership in the associated set, being of type float. Rules( ) steps through each rule in turn, looking at the associated values in InputMembership[ ]. It will then compare the two values, take the minimum, write it to the associated entry in OutputMembership[ ], along with the set type of the conclusion.

Defuzzifier( ) then takes the OutputMembership[ ] array and steps through the array, multiplying each membership value by the centroid of the associated set (this is read from a static table in the file, **OutputCentroid[ ]**), adding it to a running total for the numerator, and adding the membership value to a running total for the denominator. The numerator is divided by the denominator, and the result returned to fuzzy( ) and, thence, to the main program as the output of the fuzzy controller.

As can be seen, there are some strong points and weak points to the code. The strong point is that it would be a relatively straightforward task to set up a scripting language to automatically generate the fuzzy.c and rules.c files for any given problem. This, in fact, is what we hope to do, eventually. A major weak point, however, is the structure of the rule set. Presently, only two inputs and one output can be used. Also, neither OR nor NOT are supported. This extension bears further investigation.

A.2 SOURCE CODE

```
/*****************************************************************/
/*
/*  turret.c - Driver program for fuzzy turret controller
/*  simulation.
/*
/*
*****************************************************************/

#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "fuzzy.h"
#include "rules.h"


void main(void)
{
/* Declare file variables */
        FILE *in, *out;
        char instring[10];
        char *endptr;

/* Declare turret variables */
        float TurretPosition = 0;   /* range from -180 to
180
                                            degrees */
        float TurretVelocity = 0;   /* range from -36 to 36
                                          degrees per cycle */
        float CommandedPosition = 0;  /* range from -180 to
180
                                           degrees */
        float DeltaVelocity = 0;  /* range from -18 to 18
degrees
                                            per cycle */
        float DeltaPosition;   /* CommandedPosition-Turret-
                                  Position*/

        float Time = 0; /* Keeps track of simulated time in
tenths
                                of a second */

/* Open all required files */
        if((in = fopen("slew.in", "r")) == NULL)        {
                perror("Unable to open input file");
                exit (-1);
        }
```

```
        if((out = fopen("slew.out", "w")) == NULL)      {
                perror("Unable to open output file");
                exit (-1);
        }

/* Initialize fuzzy controller */
        init_fuzzy();

/* Begin */

        /* get first commanded position from input */
        fprintf(out, "  Time     Cmd Pos     Act Pos
                  Delta Pos    Velocity   Delta Vel\n\n");

        while (fgets(instring, 10, in) != NULL) {
                CommandedPosition = (float) strtod(instring,
                                            &endptr);
                if (endptr == instring +
                              strlen(instring)-1)       {
                        DeltaPosition = CommandedPosition-
                                        TurretPosition;
                        TurretVelocity = 0;
                        fprintf(out, "Commanded Position =
                               %12.3f\n",CommandedPosition);

                        while (fabs(DeltaPosition) >= .5  ||
                           fabs(TurretVelocity) >= .1)  {

                                DeltaVelocity =
                                   fuzzy(DeltaPosition,
                                         TurretVelocity);

                                fprintf(out,
                                   "%6.1f%12.3f%12.3f%12.3f%12.3f
                                   %12.3f\n",Time,
                                   CommandedPosition,
                                   TurretPosition,DeltaPosition,
                                   TurretVelocity,DeltaVelocity);

                                TurretPosition+=0.5*TurretVelocity;
                                DeltaPosition =
                                        CommandedPosition
                                        - TurretPosition;
                                TurretVelocity +=
                                        DeltaVelocity;
                                Time += 0.1;
                        }         /* end of while velocity &
                                  /* position != 0 */
                }       /* end of if valid number */
        }        /* end of while !eof */
```

```
        /* Close all open files */
        fclose (out);
        fclose (in);
}
```

```
/*****************************************************************/
/*
/* constants.h - File for defining constants in fuzzy system
/*
/*****************************************************************/


#define MAX_VARIABLES 5
#define MAX_SETS 10

/* Define the linguistic variables */
enum  FuzzyVariables {
        DeltaPosition = 0,
        PresentRate,
        DeltaRate,
        End };

/* Define the value sets the different linguistic variables
can occupy */

enum FuzzySets1 {
        LargeNegative = 0,
        MediumNegative,
        SmallNegative,
        Zero,
        SmallPositive,
        MediumPositive,
        LargePositive,
        End1 };
```

```
/*************************************************************/
/*
/*              fuzzy.h - Header file for fuzzy controller.
/*
/* The fuzzy controller will consist of three main parts:
/* 1) the fuzzy driver, which will get the inputs, call the
/* necessary functions, and return the defuzzified output to
/* the calling program; 2) the inference engine along with
/* its rule base; and 3) the defuzzifier.
/*
/* The only function the outside will see is the overlord.
/* It will be addressed strictly through a function call.
/*
/*************************************************************/
#include "constant.h"

void init_fuzzy (void);
float fuzzy (float, float);


/* Array to hold degree of membership in fuzzy sets of input
variables */

float InputMemberships [MAX_VARIABLES] [MAX_SETS];
```

```
/*****************************************************************/
/*
/*   fuzzy.c - Home to overlord of fuzzy controller.
/*
/*****************************************************************/

#include "constant.h"

extern float InputMemberships[MAX_VARIABLES][MAX_SETS];
extern void rules (void);
extern float defuzzifier (void);

/*****************************************************************/
/*
/*      Define the membership functions
/*
/*****************************************************************/


/* Declare the "M"embership "F"unction arrays for each input
or output variable */

float (* mf_DeltaPosition [MAX_SETS]) (float);
float (* mf_PresentRate [MAX_SETS]) (float);
float (* mf_DeltaRate [MAX_SETS]) (float);

/* Declare the array of membership function arrays */

float (*(* MembershipFunction [MAX_VARIABLES])) (float);


/**** Membership functions for DeltaPosition    ****/

/* DeltaPosition_0 () - Defines set membership on DeltaPosi-
/* tion for LargeNegative.  LargeNegative is a trapezoidal
/* function with its endpoints at -180 and -105 and its
/* roll-off points at -170 and -115. */

static float DeltaPosition_0 (float x)  {
        float low_end = -180;
        float high_end = -105;
        float low_roll = -170;
        float high_roll = -115;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if (x >= low_end && x <= low_roll) {
                y = (x-low_end) * 1/(low_roll - low_end);
                return y;
        }
```

```
        else if (x >= low_roll && x <= high_roll)       {
                return 1;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                high_roll);
                return y;
        }
}


/* DeltaPosition_1 () - Defines set membership on DeltaPosi
/* tion for MediumNegative.  MediumNegative is a triangle
/* function with its max at -86.25 and its endpoints at /*
-123.75 and -48.75. */

static float DeltaPosition_1 (float x) {
        float max_point = -86.25;
        float low_end = -123.75;
        float high_end = -48.75;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaPosition_2 () - Defines set membership on DeltaPosi
/* tion for SmallNegative.  SmallNegative is a triangle
/* function with its max at -33.75 and its endpoints at
/* -67.5 and 0. */

static float DeltaPosition_2 (float x) {
        float max_point = -33.75;
        float low_end = -67.5;
        float high_end = 0.0;

        float y;
```

```
        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaPosition_3 () - Defines set membership on DeltaPosi
/* tion for Zero.  Zero is a triangle function with its
/* max at 0 and its endpoints at -20 and 20. */

static float DeltaPosition_3 (float x) {
        float max_point = 0;
        float low_end = -20;
        float high_end = 20;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaPosition_4 () - Defines set membership on DeltaPosi
/* tion for SmallPositive.  SmallPositive is a triangle
/* function with its max at 33.75 and its endpoints at 0 and
/* 67.5. */

static float DeltaPosition_4 (float x) {
        float max_point = 33.75;
        float low_end = 0.0;
        float high_end = 67.5;
```

```
        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaPosition_5 () - Defines set membership on DeltaPosi
/* tion for MediumPositive.  MediumPositive is a triangle
/* function with its max at 86.25 and its endpoints at 48.75
/* and 123.75. */

static float DeltaPosition_5 (float x) {
        float max_point = 86.25;
        float low_end = 48.75;
        float high_end = 123.75;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaPosition_6 () - Defines set membership on DeltaPosi
/* tion for LargePositive.  LargePositive is a trapezoidal
/* function with its endpoints at 105 and 180 and its roll-
/* off points at 115 and 170. */

static float DeltaPosition_6 (float x)  {
        float low_end = 105;
```

```
        float high_end = 180;
        float low_roll = 115;
        float high_roll = 170;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if ( x >= low_end && x <= low_roll )       {
                y = (x - low_end) * 1/(low_roll - low_end);
                return y;
        }
        else if ( x >= low_roll && x <= high_roll)
                return 1;
        else    {
                y = (high_end - x) * (1/(high_end -
                                                high_roll));
                return y;
        }
}


/**** Membership functions for PresentRate ****/

/* PresentRate_0 () - Defines set membership on PresentRate
/* for LargeNegative.  LargeNegative is a trapezoidal func
/* tion with its endpoints at -36 and -21 and its roll-off
/* points at -34 and -23. */

static float PresentRate_0 (float x)    {
        float low_end = -36;
        float high_end = -21;
*       float low_roll = -34;
        float high_roll = -23;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if (x >= low_end && x < low_roll)  {
                y = (x - low_end) * (1/(low_roll -
                                                low_end));
                return y;
        }
        else if (x >= low_roll && x <= high_roll)
                return 1;
        else    {
                y = (high_end - x) * (1/(high_end -
                                                high_roll));
                return y;
        }
}
```

```
/* PresentRate_1 () - Defines set membership on PresentRate
/* for MediumNegative.  MediumNegative is a triangle func
/* tion with its max at -17.25 and its endpoints at -24.75
/* and -9.75. */

static float PresentRate_1 (float x) {
        float max_point = -17.25;
        float low_end = -24.75;
        float high_end = -9.75;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}



/* PresentRate_2 () - Defines set membership on PresentRate
/* for SmallNegative.  SmallNegative is a triangle function
/* with its max at -6.75 and its endpoints at -13.5 and
/* 0. */

static float PresentRate_2 (float x) {
        float max_point = -6.75;
        float low_end = -13.5;
        float high_end = 0.0;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
```

```
        }
}


/* PresentRate_3 () - Defines set membership on PresentRate
/* for Zero.  Zero is a triangle function with its max at 0
/* and its endpoints at -4 and 4. */

static float PresentRate_3 (float x) {
        float max_point = 0;
        float low_end = -4.0;
        float high_end = 4.0;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* PresentRate_4 () - Defines set membership on PresentRate
/* for SmallPositive.  SmallPositive is a triangle function
/* with its max at 6.75 and its endpoints at 0 and 13.5. */

static float PresentRate_4(float x) {
        float max_point = 6.75;
        float low_end = 0;
        float high_end = 13.5;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
```

```
                return y;
        }
}


/* PresentRate_5 () - Defines set membership on PresentRate
/* for MediumPositive.  MediumPositive is a triangle func
/* tion with its max at 17.25 and its endpoints at 9.75 and
/* 24.75. */

static float PresentRate_5 (float x) {
        float max_point = 17.25;
        float low_end = 9.75;
        float high_end = 24.75;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* PresentRate_6 () - Defines set membership on PresentRate
/* for LargePositive.  LargePositive is a trapezoidal func
/* tion with its endpoints at 21 and 36 and its roll-off
/* points at 23 and 34. */

static float PresentRate_6 (float x)    {
        float low_end = 21;
        float high_end = 36;
        float low_roll = 23;
        float high_roll = 34;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if (x > low_end && x < low_roll)   {
                y = (x - low_end) * 1/(low_roll-low_end);
                return y;
        }
```

```
        else if (x >= low_roll && x <= high_roll)
                return 1;
        else    {
                y = (high_end - x) * 1/(high_end -
                                                high_roll);
                return y;
        }
}



/**** Membership functions for DeltaRate ****/

/* DeltaRate_0 () - Defines set membership on DeltaRate for
/* LargeNegative.  LargeNegative is a trapezoidal function
/* with its endpoints at -18 and -10.5 and its roll-off
/* points at -17 and -11.5. */

static float DeltaRate_0 (float x)      {
        float low_end = -18;
        float high_end = -10.5;
        float low_roll = -17;
        float high_roll = -11.5;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if (x > low_end && x < low_roll)   {
                y = x * (1/(low_roll - low_end));
                return y;
        }
        else if (x >= low_roll && x <= high_roll)
                return 1;
        else    {
                y = (high_end - x) * 1/(high_end -
                                                high_roll);
                return y;
        }
}



/* DeltaRate_1 () - Defines set membership on DeltaRate for
/* MediumNegative.  MediumNegative is a triangular function
/* with its max at -8.625 and its endpoints at -12.375 and
/* -4.875. */

static float DeltaRate_1 (float x) {
        float max_point = -8.625;
        float low_end = -12.375;
        float high_end = -4.875;
```

```
        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaRate_2 () - Defines set membership on DeltaRate for
/* SmallNegative.  SmallNegative is a triangle function with
/* its max at -3.375 and its endpoints at -6.75 and -0. */

static float DeltaRate_2 (float x) {
        float max_point = -3.375;
        float low_end = -6.75;
        float high_end = 0.0;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                        max_point);
                return y;
        }
}


/* DeltaRate_3 () - Defines set membership on DeltaRate for
/* Zero.  Zero is a triangle function with its
/* max at 0 and its endpoints at -2 and 2. */

static float DeltaRate_3 (float x) {
        float max_point = 0;
        float low_end = -2.0;
```

```
        float high_end = 2.0;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                                max_point);
                return y;
        }
}


/* DeltaRate_4 () - Defines set membership on DeltaRate for
/* SmallPositive.  SmallPositive is a triangle function with
/* its max at 3.375 and its endpoints at 0 and 6.75. */

static float DeltaRate_4 (float x) {
        float max_point = 3.375;
        float low_end = 0.0;
        float high_end = 6.75;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = (high_end - x) * 1/(high_end -
                                        max_point);
                return y;
        }
}


/* DeltaRate_5 () - Defines set membership on DeltaRate for
/* MediumPositive.  MediumPositive is a triangle function
/* with its max at 8.625 and its endpoints at 4.875 and
/* 12.375. */

static float DeltaRate_5 (float x) {
```

```
        float max_point = 8.625;
        float low_end = 4.875;
        float high_end = 12.375;

        float y;

        if (x <= low_end || x >= high_end)
                return 0;
        else if (x > low_end && x <= max_point) {
                y = (x - low_end) * 1/(max_point - low_end);
                return y;
        }
        else    {
                y = x * 1/(max_point - high_end);
                return y;
        }
}


/* DeltaRate_6 () - Defines set membership on DeltaRate for
/* LargePositive.  LargePositive is a trapezoidal function
/* with its endpoints at 10.5 and 18 and its roll-off points
/* at 11.5 and 17. */

static float DeltaRate_6 (float x)      {
        float low_end = 10.5;
        float high_end = 18;
        float low_roll = 11.5;
        float high_roll = 17;

        float y;
        if (x <= low_end || x > high_end)
                return 0;
        else if (x > low_end && x < low_roll)   {
                y = (x - low_end) * 1/(low_roll-low_end);
                return y;
        }
        else if (x >= low_roll && x <= high_roll)
                return 1;
        else    {
                y = (high_end - x) * 1/(high_end -
                                                high_roll);
                return y;
        }
}

/*** Initialize fuzzy controller ***/

void init_fuzzy(void)   {
/* Assign membership functions to the appropriate arrays */
```

```
        mf_DeltaPosition[0] =   DeltaPosition_0;
        mf_DeltaPosition[1] =   DeltaPosition_1;
        mf_DeltaPosition[2] =   DeltaPosition_2;
        mf_DeltaPosition[3] =   DeltaPosition_3;
        mf_DeltaPosition[4] =   DeltaPosition_4;
        mf_DeltaPosition[5] =   DeltaPosition_5;
        mf_DeltaPosition[6] =   DeltaPosition_6;

        mf_PresentRate[0] =   PresentRate_0;
        mf_PresentRate[1] =   PresentRate_1;
        mf_PresentRate[2] =   PresentRate_2;
        mf_PresentRate[3] =   PresentRate_3;
        mf_PresentRate[4] =   PresentRate_4;
        mf_PresentRate[5] =   PresentRate_5;
        mf_PresentRate[6] =   PresentRate_6;

        mf_DeltaRate[0] =   DeltaRate_0;
        mf_DeltaRate[1] =   DeltaRate_1;
        mf_DeltaRate[2] =   DeltaRate_2;
        mf_DeltaRate[3] =   DeltaRate_3;
        mf_DeltaRate[4] =   DeltaRate_4;
        mf_DeltaRate[5] =   DeltaRate_5;
        mf_DeltaRate[6] =   DeltaRate_6;

/* Assign the individual membership function arrays to the
central array */

        MembershipFunction[DeltaPosition] =
                                        mf_DeltaPosition;
        MembershipFunction[PresentRate] = mf_PresentRate;
        MembershipFunction[DeltaRate] = mf_DeltaRate;
}



/*************************************************************/
/*
/* fuzzy() - Driver program for fuzzy logic inference
/* engine.
/* Operations:
/* 1) Will determine which of the above sets are active and
/*    to what degree.  Mark them as active and save value.
/* 2) Invoke each law in turn with values determined above.
/*    Save resulting values.
/* 3) Invoke defuzzifier and return combined value.
/*
/* Inputs:  float CmndPos, float PresRate
/* Output:  float ChangeInRate
/*************************************************************/
```

```
float fuzzy(float CmndPos, float PresRate)      {
        float ChangeInRate;

        enum FuzzySets1 i = LargeNegative;
        enum FuzzySets1 j = LargeNegative;



/*** Assign values to linguistic variables based on inputs
***/

        while (i != End1)       {
                InputMemberships [DeltaPosition] [i] =
             (* (* MembershipFunction [DeltaPosition] [i]))
                                                    (CmndPos);
                i++;
        }

        while (j != End1)       {
                InputMemberships [PresentRate] [j] =
                   (* (* MembershipFunction[PresentRate]
                                 [j]))(PresRate);
                j++;
        }

/* Apply rule set to computed membership values */
        rules();

/* Return defuzzified value */
        ChangeInRate = defuzzifier();
        return ChangeInRate;
}
```

```
/*****************************************************************/
/*
/*   rules.h - Header file for rules-handling procedures
/*
/*****************************************************************/

void rules (void);
float defuzzifier (void);
```

```
/*****************************************************************/
/*
/* rules.c - functions and structures for handling a fuzzy
rule /*  set
/*
/*****************************************************************/

#include "constant.h"
#define NumberOfRules 21

#define min(x,y) x <= y ? x : y

extern float InputMemberships[MAX_VARIABLES][MAX_SETS];


/* Array for holding the centroid of each fuzzy set on the
/* output*/
static float OutputCentroid[MAX_SETS] = {
        -14.25, /* LargeNegative */
        -8.625, /* MediumNegative */
        -3.375, /* SmallNegative */
        0.00,   /* Zero */
        3.375,  /* SmallPositive */
        8.625,  /* MediumPositive */
        14.25   /* LargePositive */
};

static struct rule      {
        enum FuzzySets1 Proposition1;
        enum FuzzySets1 Proposition2;
        enum FuzzySets1 Conclusion;
};

static struct rule RuleSet [NumberOfRules] = {
        {LargePositive, LargeNegative, LargePositive},
        {MediumPositive, MediumNegative, LargePositive},
        {MediumNegative, SmallNegative, SmallNegative},
        {SmallNegative, SmallNegative, Zero},
        {Zero, SmallNegative, SmallPositive},
        {SmallPositive, SmallNegative, MediumPositive},
        {MediumPositive, SmallNegative, LargePositive},
        {LargeNegative, Zero, SmallNegative},
        {MediumNegative, Zero, SmallNegative},
        {SmallNegative, Zero, SmallNegative},
        {Zero, Zero, Zero},
        {SmallPositive, Zero, SmallPositive},
        {MediumPositive, Zero, SmallPositive},
        {LargePositive, Zero, SmallPositive},
        {MediumNegative, SmallPositive, LargeNegative},
        {SmallNegative, SmallPositive, MediumNegative},
        {Zero, SmallPositive, SmallNegative},
```

```
        {SmallPositive, SmallPositive, Zero},
        {MediumPositive, SmallPositive, SmallPositive},
        {MediumNegative, MediumPositive, LargeNegative},
        {LargeNegative, LargePositive, LargeNegative}
};

/* Array for holding degree of membership of output of rule
set */
static struct {
        float DegreeOfMembership;
        enum FuzzySets1 set;
}  OutputMembership[NumberOfRules];



/**** rules - Using values in InputMembership array and
/**** rules from RuleSet array, compute membership in output
/**** for each individual rule and write to result array.
****/
void rules(void)        {
        int i;
        float x,y,z;
        struct rule temp;
        for (i = 0; i < NumberOfRules; i++)     {
                x = InputMemberships[(int)DeltaPosition]
                                [RuleSet[i].Proposition1];
                y = InputMemberships[(int)PresentRate]
                                [RuleSet[i].Proposition2];
                z = min(x,y);
                OutputMembership [i].DegreeOfMembership = z;
                OutputMembership [i].set =
                                        RuleSet[i].Conclusion;
        }
}


/*************************************************************/
/*
/* defuzzifier() - Will return a single value as a result
/* checking the rule set.
/*
/*************************************************************/
float defuzzifier (void)        {
        float numerator = 0;
        float denominator = 0;
        float result;
        int i;

        for (i = 0; i < NumberOfRules; i++)     {
                numerator += OutputCentroid
                    [OutputMembership[i].set] *
```

```
                    OutputMembership[i].DegreeOfMembership;
                denominator +=
                    OutputMembership[i].DegreeOfMembership;
            }
            result = (denominator == 0) ? 0 :
                                    (numerator/denominator);
            return (result);
        }
```

APPENDIX B
NUMERICAL RESULTS

APPENDIX B
NUMERICAL RESULTS

B.1 - RUN 1

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
Commanded Position =			-12.500		
0.0	-12.500	0.000	-12.500	0.000	-1.677
0.1	-12.500	0.000	-12.500	-1.677	-0.331
0.2	-12.500	-0.839	-11.661	-2.008	-0.119
0.3	-12.500	-1.843	-10.657	-2.128	-0.001
0.4	-12.500	-2.906	-9.594	-2.129	0.078
0.5	-12.500	-3.971	-8.529	-2.051	0.133
0.6	-12.500	-4.997	-7.503	-1.918	0.167
0.7	-12.500	-5.956	-6.544	-1.751	0.183
0.8	-12.500	-6.831	-5.669	-1.568	0.185
0.9	-12.500	-7.615	-4.885	-1.384	0.177
1.0	-12.500	-8.307	-4.193	-1.207	0.163
1.1	-12.500	-8.910	-3.590	-1.043	0.147
1.2	-12.500	-9.432	-3.068	-0.896	0.130
1.3	-12.500	-9.880	-2.620	-0.767	0.113
1.4	-12.500	-10.263	-2.237	-0.654	0.097
1.5	-12.500	-10.590	-1.910	-0.557	0.083
1.6	-12.500	-10.869	-1.631	-0.474	0.071
1.7	-12.500	-11.106	-1.394	-0.404	0.060
1.8	-12.500	-11.308	-1.192	-0.344	0.051
1.9	-12.500	-11.480	-1.020	-0.293	0.043
2.0	-12.500	-11.626	-0.874	-0.250	0.037
2.1	-12.500	-11.751	-0.749	-0.213	0.031
2.2	-12.500	-11.858	-0.642	-0.182	0.026
2.3	-12.500	-11.949	-0.551	-0.156	0.022
2.4	-12.500	-12.027	-0.473	-0.133	0.019
2.5	-12.500	-12.093	-0.407	-0.114	0.016
Commanded Position =			-15.000		
2.6	-15.000	-12.150	-2.850	0.000	-0.303
2.7	-15.000	-12.150	-2.850	-0.303	-0.130
2.8	-15.000	-12.302	-2.698	-0.432	-0.050
2.9	-15.000	-12.518	-2.482	-0.482	-0.007
3.0	-15.000	-12.759	-2.241	-0.489	0.019
3.1	-15.000	-13.003	-1.997	-0.470	0.033
3.2	-15.000	-13.238	-1.762	-0.437	0.040
3.3	-15.000	-13.457	-1.543	-0.397	0.042
3.4	-15.000	-13.655	-1.345	-0.355	0.041
3.5	-15.000	-13.833	-1.167	-0.314	0.039
3.6	-15.000	-13.990	-1.010	-0.275	0.035
3.7	-15.000	-14.127	-0.873	-0.240	0.032
3.8	-15.000	-14.247	-0.753	-0.208	0.028
3.9	-15.000	-14.351	-0.649	-0.180	0.025
4.0	-15.000	-14.441	-0.559	-0.155	0.021
4.1	-15.000	-14.519	-0.481	-0.134	0.019
4.2	-15.000	-14.586	-0.414	-0.115	0.016
Commanded Position =			-10.000		
4.3	10.000	-14.643	24.643	0.000	3.375

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
4.4	10.000	-14.643	24.643	3.375	0.804
4.5	10.000	-12.956	22.956	4.179	0.000
4.6	10.000	-10.867	20.867	4.179	0.000
4.7	10.000	-8.777	18.777	4.179	-0.334
4.8	10.000	-6.688	16.688	3.844	-0.580
4.9	10.000	-4.766	14.766	3.265	-0.247
5.0	10.000	-3.133	13.133	3.018	-0.270
5.1	10.000	-1.624	11.624	2.748	-0.231
5.2	10.000	-0.250	10.250	2.517	-0.173
5.3	10.000	1.009	8.991	2.344	-0.211
5.4	10.000	2.181	7.819	2.133	-0.229
5.5	10.000	3.248	6.752	1.905	-0.230
5.6	10.000	4.200	5.800	1.675	-0.220
5.7	10.000	5.037	4.963	1.456	-0.202
5.8	10.000	5.765	4.235	1.253	-0.181
5.9	10.000	6.392	3.608	1.073	-0.159
6.0	10.000	6.928	3.072	0.914	-0.137
6.1	10.000	7.385	2.615	0.776	-0.118
6.2	10.000	7.773	2.227	0.659	-0.100
6.3	10.000	8.102	1.898	0.558	-0.085
6.4	10.000	8.382	1.618	0.474	-0.072
6.5	10.000	8.619	1.381	0.402	-0.060
6.6	10.000	8.820	1.180	0.342	-0.051
6.7	10.000	8.990	1.010	0.291	-0.043
6.8	10.000	9.136	0.864	0.248	-0.036
6.9	10.000	9.260	0.740	0.211	-0.031
7.0	10.000	9.365	0.635	0.180	-0.026
7.1	10.000	9.455	0.545	0.154	-0.022
7.2	10.000	9.532	0.468	0.132	-0.019
7.3	10.000	9.598	0.402	0.113	-0.016
Commanded Position =			12.000		
7.4	12.000	9.655	2.345	0.000	0.246
7.5	12.000	9.655	2.345	0.246	0.109
7.6	12.000	9.778	2.222	0.355	0.042
7.7	12.000	9.955	2.045	0.397	0.005
7.8	12.000	10.154	1.846	0.403	-0.016
7.9	12.000	10.355	1.645	0.387	-0.027
8.0	12.000	10.549	1.451	0.360	-0.033
8.1	12.000	10.729	1.271	0.327	-0.035
8.2	12.000	10.892	1.108	0.292	-0.034
8.3	12.000	11.038	0.962	0.258	-0.032
8.4	12.000	11.167	0.833	0.226	-0.029
8.5	12.000	11.280	0.720	0.197	-0.026
8.6	12.000	11.378	0.622	0.171	-0.023
8.7	12.000	11.464	0.536	0.148	-0.020
8.8	12.000	11.538	0.462	0.128	-0.018
8.9	12.000	11.602	0.398	0.110	-0.015
Commanded Position =			180.000		
9.0	180.000	11.657	168.343	0.000	3.375

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
9.1	180.000	11.657	168.343	3.375	3.375
9.2	180.000	13.345	166.655	6.750	0.000
9.3	180.000	16.720	163.280	6.750	0.000
9.4	180.000	20.095	159.905	6.750	0.000
9.5	180.000	23.470	156.530	6.750	0.000
9.6	180.000	26.845	153.155	6.750	0.000
9.7	180.000	30.220	149.780	6.750	0.000
9.8	180.000	33.595	146.405	6.750	0.000
9.9	180.000	36.970	143.030	6.750	0.000
10.0	180.000	40.345	139.655	6.750	0.000
10.1	180.000	43.720	136.280	6.750	0.000
10.2	180.000	47.095	132.905	6.750	0.000
10.3	180.000	50.470	129.530	6.750	0.000
10.4	180.000	53.845	126.155	6.750	0.000
10.5	180.000	57.220	122.780	6.750	3.375
10.6	180.000	60.595	119.405	10.125	3.375
10.7	180.000	65.657	114.343	13.500	0.000
10.8	180.000	72.407	107.593	13.500	0.000
10.9	180.000	79.157	100.843	13.500	0.000
11.0	180.000	85.907	94.093	13.500	0.000
11.1	180.000	92.657	87.343	13.500	0.000
11.2	180.000	99.407	80.593	13.500	0.000
11.3	180.000	106.157	73.843	13.500	0.000
11.4	180.000	112.907	67.093	13.500	0.000
11.5	180.000	119.657	60.343	13.500	0.000
11.6	180.000	126.407	53.593	13.500	0.000
11.7	180.000	133.157	46.843	13.500	0.000
11.8	180.000	139.907	40.093	13.500	0.000
11.9	180.000	146.657	33.343	13.500	0.000
12.0	180.000	153.407	26.593	13.500	0.000
12.1	180.000	160.157	19.843	13.500	0.000
12.2	180.000	166.907	13.093	13.500	0.000
12.3	180.000	173.657	6.343	13.500	0.000
12.4	180.000	180.407	-0.407	13.500	0.000
12.5	180.000	187.157	-7.157	13.500	0.000
12.6	180.000	193.907	-13.907	13.500	0.000
12.7	180.000	200.657	-20.657	13.500	0.000
12.8	180.000	207.407	-27.407	13.500	0.000
12.9	180.000	214.157	-34.157	13.500	0.000
13.0	180.000	220.907	-40.907	13.500	0.000
13.1	180.000	227.657	-47.657	13.500	0.000
13.2	180.000	234.407	-54.407	13.500	-14.250
13.3	180.000	241.157	-61.157	-0.750	-2.869
13.4	180.000	240.782	-60.782	-3.619	-2.429
13.5	180.000	238.973	-58.973	-6.048	-1.752
13.6	180.000	235.949	-55.949	-7.800	-1.213
13.7	180.000	232.049	-52.049	-9.013	-0.544
13.8	180.000	227.542	-47.542	-9.557	0.000
13.9	180.000	222.764	-42.764	-9.557	0.000

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
14.0	180.000	217.986	-37.986	-9.557	0.000
14.1	180.000	213.207	-33.207	-9.557	0.000
14.2	180.000	208.429	-28.429	-9.557	0.000
14.3	180.000	203.651	-23.651	-9.557	0.000
14.4	180.000	198.872	-18.872	-9.557	0.309
14.5	180.000	194.094	-14.094	-9.248	1.398
14.6	180.000	189.470	-9.470	-7.850	2.202
14.7	180.000	185.545	-5.545	-5.648	2.750
14.8	180.000	182.721	-2.721	-2.898	1.359
14.9	180.000	181.272	-1.272	-1.539	0.699
15.0	180.000	180.503	-0.503	-0.840	0.392
15.1	180.000	180.083	-0.083	-0.448	0.225
15.2	180.000	179.859	0.141	-0.223	0.164
Commanded Position =			0.000		
15.3	0.000	179.747	-179.747	0.000	-3.375
15.4	0.000	179.747	-179.747	-3.375	-3.375
15.5	0.000	178.059	-178.059	-6.750	0.000
15.6	0.000	174.684	-174.684	-6.750	0.000
15.7	0.000	171.309	-171.309	-6.750	0.000
15.8	0.000	167.934	-167.934	-6.750	0.000
15.9	0.000	164.559	-164.559	-6.750	0.000
16.0	0.000	161.184	-161.184	-6.750	0.000
16.1	0.000	157.809	-157.809	-6.750	0.000
16.2	0.000	154.434	-154.434	-6.750	0.000
16.3	0.000	151.059	-151.059	-6.750	0.000
16.4	0.000	147.684	-147.684	-6.750	0.000
16.5	0.000	144.309	-144.309	-6.750	0.000
16.6	0.000	140.934	-140.934	-6.750	0.000
16.7	0.000	137.559	-137.559	-6.750	0.000
16.8	0.000	134.184	-134.184	-6.750	0.000
16.9	0.000	130.809	-130.809	-6.750	0.000
17.0	0.000	127.434	-127.434	-6.750	0.000
17.1	0.000	124.059	-124.059	-6.750	0.000
17.2	0.000	120.684	-120.684	-6.750	-3.375
17.3	0.000	117.309	-117.309	-10.125	-3.375
17.4	0.000	112.247	-112.247	-13.500	0.000
17.5	0.000	105.497	-105.497	-13.500	0.000
17.6	0.000	98.747	-98.747	-13.500	0.000
17.7	0.000	91.997	-91.997	-13.500	0.000
17.8	0.000	85.247	-85.247	-13.500	0.000
17.9	0.000	78.497	-78.497	-13.500	0.000
18.0	0.000	71.747	-71.747	-13.500	0.000
18.1	0.000	64.997	-64.997	-13.500	0.000
18.2	0.000	58.247	-58.247	-13.500	0.000
18.3	0.000	51.497	-51.497	-13.500	0.000
18.4	0.000	44.747	-44.747	-13.500	0.000
18.5	0.000	37.997	-37.997	-13.500	0.000
18.6	0.000	31.247	-31.247	-13.500	0.000
18.7	0.000	24.497	-24.497	-13.500	0.000

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
18.8	0.000	17.747	-17.747	-13.500	0.000
18.9	0.000	10.997	-10.997	-13.500	0.000
19.0	0.000	4.247	-4.247	-13.500	0.000
19.1	0.000	-2.503	2.503	-13.500	0.000
19.2	0.000	-9.253	9.253	-13.500	0.000
19.3	0.000	-16.003	16.003	-13.500	0.000
19.4	0.000	-22.753	22.753	-13.500	0.000
19.5	0.000	-29.503	29.503	-13.500	0.000
19.6	0.000	-36.253	36.253	-13.500	0.000
19.7	0.000	-43.003	43.003	-13.500	0.000
19.8	0.000	-49.753	49.753	-13.500	14.250
19.9	0.000	-56.503	56.503	0.750	2.878
20.0	0.000	-56.128	56.128	3.628	1.795
20.1	0.000	-54.314	54.314	5.423	0.929
20.2	0.000	-51.603	51.603	6.352	0.469
20.3	0.000	-48.427	48.427	6.821	0.000
20.4	0.000	-45.016	45.016	6.821	0.000
20.5	0.000	-41.606	41.606	6.821	0.000
20.6	0.000	-38.195	38.195	6.821	0.000
20.7	0.000	-34.785	34.785	6.821	0.000
20.8	0.000	-31.374	31.374	6.821	0.000
20.9	0.000	-27.963	27.963	6.821	0.000
21.0	0.000	-24.553	24.553	6.821	0.000
21.1	0.000	-21.142	21.142	6.821	0.000
21.2	0.000	-17.732	17.732	6.821	-0.599
21.3	0.000	-14.321	14.321	6.222	-1.353
21.4	0.000	-11.210	11.210	4.869	-1.922
21.5	0.000	-8.776	8.776	2.947	-0.488
21.6	0.000	-7.303	7.303	2.458	-0.422
21.7	0.000	-6.073	6.073	2.036	-0.356
21.8	0.000	-5.055	5.055	1.680	-0.296
21.9	0.000	-4.215	4.215	1.384	-0.244
22.0	0.000	-3.524	3.524	1.140	-0.199
22.1	0.000	-2.954	2.954	0.941	-0.162
22.2	0.000	-2.483	2.483	0.779	-0.132
22.3	0.000	-2.094	2.094	0.647	-0.108
22.4	0.000	-1.771	1.771	0.539	-0.088
22.5	0.000	-1.501	1.501	0.451	-0.072
22.6	0.000	-1.276	1.276	0.379	-0.060
22.7	0.000	-1.086	1.086	0.319	-0.049
22.8	0.000	-0.927	0.927	0.270	-0.041
22.9	0.000	-0.792	0.792	0.229	-0.034
23.0	0.000	-0.678	0.678	0.194	-0.029
23.1	0.000	-0.580	0.580	0.166	-0.024
23.2	0.000	-0.498	0.498	0.141	-0.021
23.3	0.000	-0.427	0.427	0.121	-0.017
23.4	0.000	-0.367	0.367	0.103	-0.015

Commanded Position = 0.000

B.2 - RUN 2

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
Commanded Position =			-12.500		
0.0	-12.500	0.000	-12.500	0.000	-1.677
0.1	-12.500	0.000	-12.500	-1.677	-0.331
0.2	-12.500	-0.839	-11.661	-2.008	-0.119
0.3	-12.500	-1.843	-10.657	-2.128	-0.001
0.4	-12.500	-2.906	-9.594	-2.129	0.078
0.5	-12.500	-3.971	-8.529	-2.051	0.133
0.6	-12.500	-4.997	-7.503	-1.918	0.167
0.7	-12.500	-5.956	-6.544	-1.751	0.183
0.8	-12.500	-6.831	-5.669	-1.568	0.185
0.9	-12.500	-7.615	-4.885	-1.384	0.177
1.0	-12.500	-8.307	-4.193	-1.207	0.163
1.1	-12.500	-8.910	-3.590	-1.043	0.147
1.2	-12.500	-9.432	-3.068	-0.896	0.130
1.3	-12.500	-9.880	-2.620	-0.767	0.113
1.4	-12.500	-10.263	-2.237	-0.654	0.097
1.5	-12.500	-10.590	-1.910	-0.557	0.083
1.6	-12.500	-10.869	-1.631	-0.474	0.071
1.7	-12.500	-11.106	-1.394	-0.404	0.060
1.8	-12.500	-11.308	-1.192	-0.344	0.051
1.9	-12.500	-11.480	-1.020	-0.293	0.043
2.0	-12.500	-11.626	-0.874	-0.250	0.037
2.1	-12.500	-11.751	-0.749	-0.213	0.031
2.2	-12.500	-11.858	-0.642	-0.182	0.026
2.3	-12.500	-11.949	-0.551	-0.156	0.022
2.4	-12.500	-12.027	-0.473	-0.133	0.019
2.5	-12.500	-12.093	-0.407	-0.114	0.016
Commanded Position =			-15.000		
2.6	-15.000	-12.150	-2.850	0.000	-0.303
2.7	-15.000	-12.150	-2.850	-0.303	-0.130
2.8	-15.000	-12.302	-2.698	-0.432	-0.050
2.9	-15.000	-12.518	-2.482	-0.482	-0.007
3.0	-15.000	-12.759	-2.241	-0.489	0.019
3.1	-15.000	-13.003	-1.997	-0.470	0.033
3.2	-15.000	-13.238	-1.762	-0.437	0.040
3.3	-15.000	-13.457	-1.543	-0.397	0.042
3.4	-15.000	-13.655	-1.345	-0.355	0.041
3.5	-15.000	-13.833	-1.167	-0.314	0.039
3.6	-15.000	-13.990	-1.010	-0.275	0.035
3.7	-15.000	-14.127	-0.873	-0.240	0.032
3.8	-15.000	-14.247	-0.753	-0.208	0.028
3.9	-15.000	-14.351	-0.649	-0.180	0.025
4.0	-15.000	-14.441	-0.559	-0.155	0.021
4.1	-15.000	-14.519	-0.481	-0.134	0.019
4.2	-15.000	-14.586	-0.414	-0.115	0.016
Commanded Position =			10.000		
4.3	10.000	-14.643	24.643	0.000	3.375

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
4.4	10.000	-14.643	24.643	3.375	0.804
4.5	10.000	-12.956	22.956	4.179	0.000
4.6	10.000	-10.867	20.867	4.179	0.000
4.7	10.000	-8.777	18.777	4.179	-0.334
4.8	10.000	-6.688	16.688	3.844	-0.580
4.9	10.000	-4.766	14.766	3.265	-0.247
5.0	10.000	-3.133	13.133	3.018	-0.270
5.1	10.000	-1.624	11.624	2.748	-0.231
5.2	10.000	-0.250	10.250	2.517	-0.173
5.3	10.000	1.009	8.991	2.344	-0.211
5.4	10.000	2.181	7.819	2.133	-0.229
5.5	10.000	3.248	6.752	1.905	-0.230
5.6	10.000	4.200	5.800	1.675	-0.220
5.7	10.000	5.037	4.963	1.456	-0.202
5.8	10.000	5.765	4.235	1.253	-0.181
5.9	10.000	6.392	3.608	1.073	-0.159
6.0	10.000	6.928	3.072	0.914	-0.137
6.1	10.000	7.385	2.615	0.776	-0.118
6.2	10.000	7.773	2.227	0.659	-0.100
6.3	10.000	8.102	1.898	0.558	-0.085
6.4	10.000	8.382	1.618	0.474	-0.072
6.5	10.000	8.619	1.381	0.402	-0.060
6.6	10.000	8.820	1.180	0.342	-0.051
6.7	10.000	8.990	1.010	0.291	-0.043
6.8	10.000	9.136	0.864	0.248	-0.036
6.9	10.000	9.260	0.740	0.211	-0.031
7.0	10.000	9.365	0.635	0.180	-0.026
7.1	10.000	9.455	0.545	0.154	-0.022
7.2	10.000	9.532	0.468	0.132	-0.019
7.3	10.000	9.598	0.402	0.113	-0.016
Commanded Position =		12.000			
7.4	12.000	9.655	2.345	0.000	0.246
7.5	12.000	9.655	2.345	0.246	0.109
7.6	12.000	9.778	2.222	0.355	0.042
7.7	12.000	9.955	2.045	0.397	0.005
7.8	12.000	10.154	1.846	0.403	-0.016
7.9	12.000	10.355	1.645	0.387	-0.027
8.0	12.000	10.549	1.451	0.360	-0.033
8.1	12.000	10.729	1.271	0.327	-0.035
8.2	12.000	10.892	1.108	0.292	-0.034
8.3	12.000	11.038	0.962	0.258	-0.032
8.4	12.000	11.167	0.833	0.226	-0.029
8.5	12.000	11.280	0.720	0.197	-0.026
8.6	12.000	11.378	0.622	0.171	-0.023
8.7	12.000	11.464	0.536	0.148	-0.020
8.8	12.000	11.538	0.462	0.128	-0.018
8.9	12.000	11.602	0.398	0.110	-0.015
Commanded Position =		180.000			
9.0	180.000	11.657	168.343	0.000	3.375

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
9.1	180.000	11.657	168.343	3.375	3.375
9.2	180.000	13.345	166.655	6.750	0.000
9.3	180.000	16.720	163.280	6.750	0.000
9.4	180.000	20.095	159.905	6.750	0.000
9.5	180.000	23.470	156.530	6.750	0.000
9.6	180.000	26.845	153.155	6.750	0.000
9.7	180.000	30.220	149.780	6.750	0.000
9.8	180.000	33.595	146.405	6.750	0.000
9.9	180.000	36.970	143.030	6.750	0.000
10.0	180.000	40.345	139.655	6.750	0.000
10.1	180.000	43.720	136.280	6.750	0.000
10.2	180.000	47.095	132.905	6.750	0.000
10.3	180.000	50.470	129.530	6.750	0.000
10.4	180.000	53.845	126.155	6.750	0.000
10.5	180.000	57.220	122.780	6.750	0.000
10.6	180.000	60.595	119.405	6.750	0.000
10.7	180.000	63.970	116.030	6.750	0.000
10.8	180.000	67.345	112.655	6.750	0.000
10.9	180.000	70.720	109.280	6.750	0.000
11.0	180.000	74.095	105.905	6.750	0.000
11.1	180.000	77.470	102.530	6.750	0.000
11.2	180.000	80.845	99.155	6.750	0.000
11.3	180.000	84.220	95.780	6.750	0.000
11.4	180.000	87.595	92.405	6.750	0.000
11.5	180.000	90.970	89.030	6.750	0.000
11.6	180.000	94.345	85.655	6.750	0.000
11.7	180.000	97.720	82.280	6.750	0.000
11.8	180.000	101.095	78.905	6.750	0.000
11.9	180.000	104.470	75.530	6.750	0.000
12.0	180.000	107.845	72.155	6.750	0.000
12.1	180.000	111.220	68.780	6.750	0.000
12.2	180.000	114.595	65.405	6.750	0.000
12.3	180.000	117.970	62.030	6.750	0.000
12.4	180.000	121.345	58.655	6.750	0.000
12.5	180.000	124.720	55.280	6.750	0.000
12.6	180.000	128.095	51.905	6.750	0.000
12.7	180.000	131.470	48.530	6.750	0.000
12.8	180.000	134.845	45.155	6.750	0.000
12.9	180.000	138.220	41.780	6.750	0.000
13.0	180.000	141.595	38.405	6.750	0.000
13.1	180.000	144.970	35.030	6.750	0.000
13.2	180.000	148.345	31.655	6.750	0.000
13.3	180.000	151.720	28.280	6.750	0.000
13.4	180.000	155.095	24.905	6.750	0.000
13.5	180.000	158.470	21.530	6.750	0.000
13.6	180.000	161.845	18.155	6.750	-0.494
13.7	180.000	165.220	14.780	6.256	-1.260
13.8	180.000	168.348	11.652	4.996	-1.847
13.9	180.000	170.846	9.154	3.149	-0.736

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
14.0	180.000	172.420	7.580	2.413	-0.373
14.1	180.000	173.626	6.374	2.040	-0.327
14.2	180.000	174.647	5.353	1.713	-0.281
14.3	180.000	175.503	4.497	1.432	-0.238
14.4	180.000	176.219	3.781	1.194	-0.199
14.5	180.000	176.816	3.184	0.996	-0.165
14.6	180.000	177.314	2.686	0.831	-0.137
14.7	180.000	177.729	2.271	0.694	-0.113
14.8	180.000	178.076	1.924	0.581	-0.093
14.9	180.000	178.367	1.633	0.488	-0.077
15.0	180.000	178.611	1.389	0.411	-0.064
15.1	180.000	178.817	1.183	0.347	-0.053
15.2	180.000	178.990	1.010	0.294	-0.045
15.3	180.000	179.137	0.863	0.249	-0.037
15.4	180.000	179.262	0.738	0.212	-0.031
15.5	180.000	179.367	0.633	0.180	-0.026
15.6	180.000	179.458	0.542	0.154	-0.022
15.7	180.000	179.535	0.465	0.132	-0.019
15.8	180.000	179.600	0.400	0.113	-0.016
Commanded Position =			0.000		
15.9	0.000	179.657	-179.65	0.000	-3.375
16.0	0.000	179.657	-179.65	-3.375	-3.375
16.1	0.000	177.969	-177.969	-6.750	0.000
16.2	0.000	174.594	-174.594	-6.750	0.000
16.3	0.000	171.219	-171.219	-6.750	0.000
16.4	0.000	167.844	-167.844	-6.750	0.000
16.5	0.000	164.469	-164.469	-6.750	0.000
16.6	0.000	161.094	-161.094	-6.750	0.000
16.7	0.000	157.719	-157.719	-6.750	0.000
16.8	0.000	154.344	-154.344	-6.750	0.000
16.9	0.000	150.969	-150.969	-6.750	0.000
17.0	0.000	147.594	-147.594	-6.750	0.000
17.1	0.000	144.219	-144.219	-6.750	0.000
17.2	0.000	140.844	-140.844	-6.750	0.000
17.3	0.000	137.469	-137.469	-6.750	0.000
17.4	0.000	134.094	-134.094	-6.750	0.000
17.5	0.000	130.719	-130.719	-6.750	0.000
17.6	0.000	127.344	-127.344	-6.750	0.000
17.7	0.000	123.969	-123.969	-6.750	0.000
17.8	0.000	120.594	-120.594	-6.750	0.000
17.9	0.000	117.219	-117.219	-6.750	0.000
18.0	0.000	113.844	-113.844	-6.750	0.000
18.1	0.000	110.469	-110.469	-6.750	0.000
18.2	0.000	107.094	-107.094	-6.750	0.000
18.3	0.000	103.719	-103.719	-6.750	0.000
18.4	0.000	100.344	-100.344	-6.750	0.000
18.5	0.000	96.969	-96.969	-6.750	0.000
18.6	0.000	93.594	-93.594	-6.750	0.000
18.7	0.000	90.219	-90.219	-6.750	0.000

Time	Cmd Pos	Act Pos	Delta Pos	Vel	Delta Vel
18.8	0.000	86.844	-86.844	-6.750	0.000
18.9	0.000	83.469	-83.469	-6.750	0.000
19.0	0.000	80.094	-80.094	-6.750	0.000
19.1	0.000	76.719	-76.719	-6.750	0.000
19.2	0.000	73.344	-73.344	-6.750	0.000
19.3	0.000	69.969	-69.969	-6.750	0.000
19.4	0.000	66.594	-66.594	-6.750	0.000
19.5	0.000	63.219	-63.219	-6.750	0.000
19.6	0.000	59.844	-59.844	-6.750	0.000
19.7	0.000	56.469	-56.469	-6.750	0.000
19.8	0.000	53.094	-53.094	-6.750	0.000
19.9	0.000	49.719	-49.719	-6.750	0.000
20.0	0.000	46.344	-46.344	-6.750	0.000
20.1	0.000	42.969	-42.969	-6.750	0.000
20.2	0.000	39.594	-39.594	-6.750	0.000
20.3	0.000	36.219	-36.219	-6.750	0.000
20.4	0.000	32.844	-32.844	-6.750	0.000
20.5	0.000	29.469	-29.469	-6.750	0.000
20.6	0.000	26.094	-26.094	-6.750	0.000
20.7	0.000	22.719	-22.719	-6.750	0.000
20.8	0.000	19.344	-19.344	-6.750	0.183
20.9	0.000	15.969	-15.969	-6.567	1.008
21.0	0.000	12.686	-12.686	-5.559	1.664
21.1	0.000	9.906	-9.906	-3.895	1.898
21.2	0.000	7.959	-7.959	-1.997	0.160
21.3	0.000	6.960	-6.960	-1.837	0.182
21.4	0.000	6.041	-6.041	-1.656	0.188
21.5	0.000	5.214	-5.214	-1.468	0.183
21.6	0.000	4.480	-4.480	-1.284	0.171
21.7	0.000	3.838	-3.838	-1.113	0.155
21.8	0.000	3.281	-3.281	-0.958	0.138
21.9	0.000	2.802	-2.802	-0.820	0.120
22.0	0.000	2.392	-2.392	-0.700	0.104
22.1	0.000	2.042	-2.042	-0.597	0.089
22.2	0.000	1.743	-1.743	-0.508	0.076
22.3	0.000	1.489	-1.489	-0.432	0.064
22.4	0.000	1.273	-1.273	-0.368	0.055
22.5	0.000	1.089	-1.089	-0.313	0.046
22.6	0.000	0.933	-0.933	-0.267	0.039
22.7	0.000	0.799	-0.799	-0.228	0.033
22.8	0.000	0.685	-0.685	-0.195	0.028
22.9	0.000	0.588	-0.588	-0.166	0.024
23.0	0.000	0.505	-0.505	-0.142	0.020
23.1	0.000	0.433	-0.433	-0.122	0.017
23.2	0.000	0.372	-0.372	-0.105	0.015

Commanded Position = 0.000

INITIAL DISTRIBUTION LIST

	Copies
U.S. Army Materiel System Analysis Activity ATTN: AMXSY–MP (Herbert Cohen) Aberdeen Proving Ground, MD 21005	1
IIT Research Institute ATTN: GACIAC 10 W. 35th Street Chicago, IL 60616	1
AMSMI–RD	1
AMSMI–RD–CS–R	15
AMSMI–RD–CS–T	1
AMSMI–GC–IP, Mr. Fred H. Bush	1
SFAE–AD–ATA–SE, Mr. Carter	2
AMSMI–RD–GC, Mr. Jacobs	1
AMSMI–RD–GC–S, Mr. Scheiman	1